이미지 와핑을 이용한 실시간 그림자 생성 기법
(Real-Time Shadow Generation using Image Warping)

강 병 권 † 임 인 성 ††
(Byungkwon Kang) (Insung Ihm)

요 약 컴퓨터 그래픽스에서 그림자는 장면의 사실성을 높이기 위하여 매우 중요한 요소이다. 전경을 렌더링 할 때 그림자의 모양이나 위치를 정확하게 나타내는 것도 중요하지만, 실제 세계에서 흔히 볼 수 있는 면적을 가지는 광원에 의한 부드러운 그림자를 효과적으로 표현하는 것도 중요하다. 그러나 현존하는 대부분의 그림자 생성 기법들은 사실적인 그림자를 실시간으로 생성해 내기에 어려움이 많다. 기존에 제안된 영상 기반 렌더링 기법을 이용하면 실시간으로 그림자를 생성해 내는데 유용하게 사용될 수가 있다. 하지만 이러한 방법에서는 일반적으로 그림자 지도의 크기가 지나치게 방대해지기 때문에, 텍스춰 메모리에 올리기에 무리가 따르게 되므로, 효율적인 압축기법이 필요하다는 단점이 있다. 이러한 그림자 지도의 크기와 압축의 부담으로 인해, 다양한 물체의 움직임을 표현하거나, 부드러운 그림자로의 확장에 어려움이 있을 수 있다. 이러한 점을 해결하고자, 본 논문에서는 영상 기반 렌더링 기법을 적용한 방법에 이미지 와핑 기법을 응용하여 그림자 지도의 크기를 대폭 줄일 수 있는 방법을 제안하고자 한다. 이 방법에서는 물체가 움직이는 범위와 상관없이 매우 적은 개수의 그림자 지도만으로 그림자를 만들어 낼 수 있기 때문에, 부드러운 그림자를 만들기 위한 방법으로 쉽게 적용할 수 있다. 이 논문에서 개발한 기법은 3차원 게임이나 가상 현실 등 관련 분야에서 사실적인 영상을 실시간으로 생성해 내는 데 유용하게 쓰일 수 있을 것이다.

키워드 : 그림자, 부드러운 그림자, 영상 기반 렌더링, 이미지 와핑, 텍스춰 매핑

Abstract Shadows are important elements in producing a realistic image. Generation of exact shapes and positions of shadows is essential in rendering since it provides users with visual cues on the scene. It is also very important to be able to create soft shadows resulted from area light sources since they increase the visual realism drastically. In spite of their importance, the existing shadow generation algorithms still have some problems in producing realistic shadows in real-time. While image-based rendering techniques can often be effectively applied to real-time shadow generation, such techniques usually demand so large memory space for storing preprocessed shadow maps. An effective compression method can help in reducing memory requirement, only at the additional decoding costs. In this paper, we propose a new image-based shadow generation method based on image warping. With this method, it is possible to generate realistic shadows using only small sizes of pre-generated shadow maps, and is easy to extend to soft shadow generation. Our method will be efficiently used for generating realistic scenes in many real-time applications such as 3D games and virtual reality systems.

Key words : Shadow, soft shadow, image-based rendering, image warping, texture mapping

1. 서 론

최근의 급격한 컴퓨터 하드웨어의 발달은 과거에는 생각도 못했던 수많은 작업들을 컴퓨터를 통해 매우 빠르게 수행하여 수준 높은 결과물을 얻을 수 있게 해준다. 이러한 하드웨어의 발달은 컴퓨터 그래픽스 분야에도 영향을 주어 왔다. 예를 들어, 이제는 범용 PC에서도 고가의 워크스테이션에서 가능했던 3차원 영상을 실시간으로 얻을 수 있으며, 이로 인해 비디오 게임 분야

· 본 논문은 정보통신부의 2001년도 대학기초연구지원사업의 연구 결과의 일부임.

† 비 회 원 : 서강대학교 컴퓨터학과
wormkbk@grmanet.sogang.ac.kr
†† 종신회원 : 서강대학교 컴퓨터학과
ihm@sogang.ac.kr

논문접수 : 2001년 7월 9일
심사완료 : 2002년 2월 20일

에서도 3차원 게임에 대한 관심이 높아지고 있다. 그러나 일반적인 3차원 그래픽스 가속기에서 처리할 수 있는 고급 렌더링 기법에 한계가 있기 때문에 실제와 같은 사실적인 영상을 얻어내기에는 아직 부족함이 있는 것 또한 사실이다. 최근에는 꾸준히 발전하고 있는 OpenGL이나 Direct3D와 같은 3차원 그래픽스 라이브러리들과 이러한 것들을 가속해주는 하드웨어들을 이용한 실시간 렌더링 분야가 각광을 받고 있다. 이러한 라이브러리와 하드웨어를 이용하면 상당히 사실감 있는 영상을 얻어낼 수 있지만, 각각 그 기능에 한계가 있어서 부드러운 그림자와 같은 여러 가지의 고급 효과들을 용이하게 얻어내지는 못한다. 3차원 컴퓨터 그래픽스에서 사실적인 영상을 얻어내기 위해서 그림자는 필수적인 요소이지만, 현존하는 3차원 그래픽스 라이브러리나 3D 가속 하드웨어들에게 부드러운 그림자처럼 사실적인 그림자를 생성하게 해주는 기능과 그것을 가속해주는 기능은 제공되지 못하고 있다. 이러한 문제점은 자연스러운 그림자를 가지는 3차원 영상을 실시간으로 얻어내는 데 큰 장애물이 되고 있다.

지금까지 그래픽스 하드웨어를 이용한 여러 그림자 생성 알고리즘들이 개발되어왔다. Crow는 한 광원이 전경에 대해 그림자를 지게 하는 영역을 그림자 볼륨이라고 하는 영역으로 구분하고, 그 안에 들어오는 물체에 대하여 그림자를 지게 하는 방법으로 그림자를 생성하였다[1]. 이 방법은 OpenGL과 같은 그래픽스 라이브러리를 사용하여 구현할 수 있는데, 간단한 물체를 이용한 전경의 경우 빠르게 그림자가 생성되지만, 전경의 복잡도에 따라 그림자의 생성 속도가 느려지는 단점이 있다.

그림자는 광원의 시점에서 보았을 때 보이지 않는 부분에는 빛이 도달하지 않기 때문에 생기는 것이다. 그렇기 때문에 광원에서 전경의 각 지점에 대한 가시성 정보를 가지고 있다면 그 정보를 이용해 그림자를 지게 만들 수 있다. Z-버퍼 알고리즘은 이런 가시성을 깊이 버퍼를 이용하여 미리 계산해 놓음으로써 가시성 정보를 만드는데 드는 시간을 줄이는 방법이다. Z-버퍼 알고리즘은 먼저 광원의 시점에서 전경을 렌더링하여 광원에서의 전경에 대한 깊이 정보를 저장한 다음, 시점에서 렌더링을 할 때에는 이 깊이 정보를 이용하여 실제 전경의 깊이와 비교해서 그림자를 지게 하는 방법이다[2]. 이렇게 생성된 깊이 정보를 그림자 지도(shadow map)이라고 하고 이러한 방법을 그림자 지도 기법이라고도 한다.

그림자 지도 기법은 텍스춰 매핑 하드웨어를 이용하여 빠르게 가속할 수 있다[3]. Segal 등은 이 논문에서 텍스춰 매핑 기법을 이용하여 텍스춰를 전경에 투영시키는 방법을 소개하고, 이 방법을 이용하여 스폿 광원(spotlight)효과나 그림자 효과를 만들 수 있다고 하였다. 이러한 방법은 일단 그림자 텍스춰만 미리 생성해 놓으면 현존하는 어떤 알고리즘보다 빠른 속도로 그림자를 생성해 낼 수 있다. 하지만, 광원이나 물체가 움직이는 경우에는 그 속도가 매우 느려지는 단점이 있다. 영상 기반 렌더링을 응용하여 물체나 광원이 움직였을 경우의 그림자 지도까지 미리 생성하여 놓는다면 물체가 움직이는 경우에도 빠르게 그림자를 생성할 수 있다[4]. 하지만, 그러한 경우 많은 수의 그림자 텍스춰를 만들어야 하기 때문에 그림자 지도의 크기가 지나치게 커져서 텍스춰 메모리의 용량이 감당하기 힘들어지게 된다. 따라서 [4]에서는 이렇게 방대한 양의 그림자 지도들을 웨이블릿 기반의 압축 기법[23]으로 압축하여 사용하였다.

위의 그림자 지도 기법들로는 그림자 지도가 미리 만들어진다면 매우 빠른 속도로 그림자를 생성할 수 있다. 그러나 이러한 방법들은 점 광원에 대한 그림자로써 부드럽지 못하고 경계가 뚜렷한 그림자(hard shadow)만들게 된다. 그러나 실제 세계에서는 이러한 그림자보다는 면적을 가지는 광원에 의한 경계가 부드러운 그림자(soft shadow)가 생기는 경우가 대부분이다. 그렇기 때문에 장면이 좀 더 사실적으로 보이기 위해서는 이러한 부드러운 그림자의 생성이 필수적이다.

Heckbert 등은 면적을 갖는 광원에 대해 광원을 여러 개의 점 광원으로 나누어 샘플링 하고 이렇게 생긴 그림자들을 누적버퍼(accumulation buffer)를 이용하여 평균내는 방법으로 부드러운 그림자를 생성하였다[6]. 이 방법은 광원의 샘플링을 많이 할수록 그림자가 부드러워지지만, 그만큼 속도가 느려지게 되고, 반대로 샘플링을 적게 하면 속도는 빨라지지만, 그림자의 경계에 계단현상이 생기는 문제점이 있다. Soler 등은 이런 문제점을 해결하기 위하여 콘볼루션(convolution) 연산을 이용하여 그림자를 부드럽게 만드는 방법을 제안하였다[7]. 그러나 두 방법 모두 기존의 그림자 지도 기법과 같이 광원이 움직이는 경우에는 그림자 지도를 새로 만들어야 하기 때문에 속도가 느려지는 단점을 가지고 있다.

이 논문에서는 기존의 영상 기반 렌더링 기법이 지나치게 큰 텍스춰 메모리를 요구하는 문제점이 있었기 때문에 이미지 와핑을 이용하여 최소한의 그림자 지도만으로 모든 위치의 그림자를 빠르게 생성해 낼 수 있는 방법을 제안하고자 한다. 즉, 중간 단계의 그림자를 만들어 낼 수 있는 최소한의 그림자 지도만을 미리 생성하여 텍스춰 메모리에 올리고, 그것들을 이용하여 이미지 와핑을 함으로써 그 사이의 위치에 대한 그림자들을 생성하고자 하였다.

본 논문의 2장에서는 기본적인 그림자 지도 기법의 개념과 텍스춰 매핑을 이용하여 그림자를 생성하는 방법에 대하여 설명하고, 3장에서는 이미지 와핑을 그림자

지도 기법에 적용하여 향상시킨 그림자 지도 기법에 대하여 설명한다. 4장에서는 이 논문에서 제안한 기법의 실제 구현과 실험 결과에 대하여 설명하고 마지막으로 5장에서 결론을 맺는다.

2. 그림자 지도 기법

2.1 기본 알고리즘

물체들의 깊이 정보를 이용한 그림자 생성 기법은 Williams에 의해 제안되었다[2]. 이 기법은 먼저 광원에서 전경을 렌더링 하여 광원과 물체들 사이의 거리 정보를 저장한 다음 실제 렌더링 과정에서 시점에서 렌더링 된 전경의 거리 정보와 비교하게 된다. 그림 1은 전처리 과정에서 만들어진 깊이 정보로써, 이것을 그림자 지도라 한다. 두 번째 과정에서 실제 렌더링을 할 때, 각 픽셀의 깊이 값을 광원을 기준으로 하는 좌표계로 변환하여 앞의 그림자 지도의 깊이 값과 비교한다. 만약 실제 렌더링하는 물체의 깊이 값이 더 큰 경우, 그 지점과 광원 사이에 빛을 가로막는 물체가 있는 것이므로 그 지점에 그림자를 지게 하면 되는 것이다. 그림 2는 이러한 그림자 지도 기법의 기본적인 구조를 보여주는 그림이다.

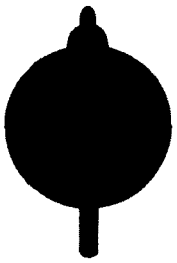


그림 1 광원의 시점에서 렌더링한 깊이 정보

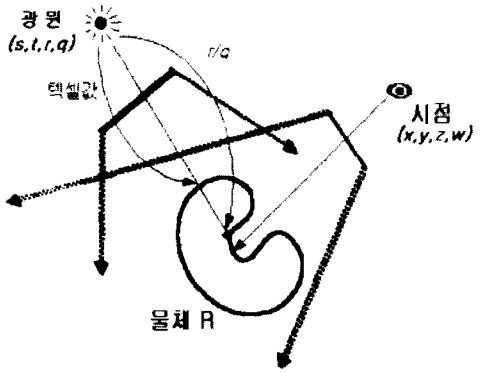


그림 2 그림자 지도 기법의 구조

2.2 텍스춰 매핑을 이용한 그림자 지도 기법

기존의 그림자 지도 기법은 그 구현이 용이하고, 정확한 그림자를 생성해 줄 수 있는 장점이 있지만, 이미지 평면의 각 픽셀 별로 그 깊이 정보와 그림자 지도의 값을 비교하는 데에는 매우 많은 계산량을 필요로 하기 때문에 그 생성 속도가 느린 단점이 있다. Segal 등은 이러한 문제점을 텍스춰 맵핑을 이용 하드웨어 가속으로 해결할 수 있는 방법을 제안하였다[3].

2.2.1 투영 텍스춰

투영 텍스춰(projective texture)는 미리 정의된 텍스춰를 영사기가 벽에 영상을 비추듯이 물체위로 투영시키는 방법이다. 이 기법은 OpenGL에서 제공하는 텍스춰 좌표계의 r, q좌표와 텍스춰 행렬 스택(texture matrix stack), 그리고 자동 텍스춰 좌표 생성 함수인 glTexGen() 함수를 이용하여 구현할 수 있다. 텍스춰를 전경에 투영시키는 작업은 텍스춰 행렬 스택에 투영 행렬을 계산하여 저장해 놓고, 자동 텍스춰 좌표 생성 함수를 통해 생성한 텍스춰의 좌표를 그대로 화면에 투영시키는 방법으로 이루어진다. 이를 위해서 텍스춰 행렬 스택에 적절한 내용의 변환 행렬을 올려주어야 하는데, 이와 같은 작업은 다음과 같은 과정을 통해 수행할 수 있다.

1. 전경에 투영의 방향을 맞추는 모델뷰 변환(model-view transform) 과정.
2. 투영 변환(projection transform) 과정(여기에서는 원근변환 혹은 직교 변환을 사용할 수 있다.).
3. 가까운 절단 평면(near clipping plane)에 대한 축소 및 이동 변환을 통하여 텍스춰 좌표계로 맞추는 과정.

첫 번째 과정에서 모델뷰 변환은 gluLookAt() 함수를 이용해서 설정할 수 있고, 두 번째의 투영변환은 glFrustum() 혹은 gluPerspective(), glOrtho() 등의 함수로 설정할 수 있다. 이러한 변환을 통해 광원을 시점으로 하는 뷰 볼륨을 설정할 수 있고, 여기에 미리 정의된 텍스춰 이미지를 가까운 절단 평면으로 대체하면 된다. 즉, 광원의 위치에서 텍스춰를 통해 전경을 보고 있는 것이라 할 수 있다.

여기에서 투영 변환은 눈 좌표계(Eye Coordinates)를 정규 디바이스 좌표계(Normalized Device Coordinates)로 변환하는 과정인데, 텍스춰 좌표계는 0과 1 사이의 구간에서 정의되는 반면, 정규 디바이스 좌표계는 -1부터 1 사이의 값으로 정의가 되기 때문에 텍스춰를 가까운 절단 평면에 맞추기 위해서는 크기와 위치를 변화시킬 필요가 있다. 그래서 정규 디바이스 좌표계로 되어있는 절단 평면을 s와 t좌표에 대해 반만큼 축소하고, 텍스춰의 중심에 맞추기 위해 0.5씩 대각선 방향으로 이

동시켜야 한다. 이러한 과정이 앞에서 언급한 세 번째 과정이다. 이러한 작업은 다음과 같은 방법으로 Open GL로 구현할 수 있다.

1. 행렬 스택을 텍스춰 행렬 스택으로 변환(glMatrix Mode(GL_TEXTURE);)
2. 텍스춰 행렬 스택을 초기화(glLoadIdentity();)
3. 중심을 맞추기 위한 이동(glTranslatef(0.5, 0.5, 0.0);)
4. 크기를 맞추기 위한 축소(glScalef(0.5, 0.5, 1.0);)
5. 투영 변환 설정(glFrustum(*); 혹은 gluPerspective(*);)
6. 모델뷰 변환 설정 (gluLookAt(*);)

2.2.2 투영 텍스춰를 이용한 그림자 지도 기법

투영 텍스춰 기법을 이용하면, 스폿 광원(spotlight) 효과나 영사기, 그림자와 같은 여러 가지 표현을 할 수 있다. 이러한 방법은 투영시킬 텍스춰만 미리 만들어 놓으면 시점의 변화나 전경의 복잡도와 상관없이 매우 빠른 속도로 영상을 만들어 낼 수 있다. 투영 텍스춰 기법은 Z-버퍼 알고리즘을 이용한 그림자를 매우 빠르게 만들어 낼 수 있다. 그림 3은 이러한 투영 텍스춰 기법을 이용한 영사기 효과를 낸 그림이다.

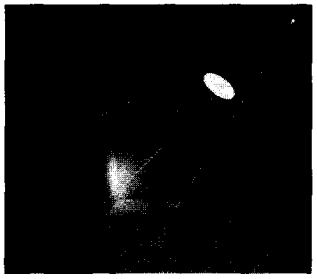


그림 3 투영 텍스춰 기법을 이용한 영사기 효과

이 방법에서는 우선 광원의 시점에서 전경을 렌더링을 한다. 이 과정에서 만들어지는 그림자 지도는 광원에서 본 전경에서 각 물체들의 깊이 정보를 저장한다. 이때, 깊이 정보는 0에서 1사이의 실수 값으로 계산되기 때문에 텍스춰의 형식에 맞추기 위해서 0에서 255사이의 값으로 바꿔주어야 한다. 이렇게 생성된 그림자 텍스춰는 투영 텍스춰 기법을 통해 전경에 투영되는데, 이 투영 과정에서 이미지 평면의 각 픽셀에 대하여 해당되는 텍스춰의 텍셀 즉, 광원에서의 깊이 정보와 텍스춰 좌표의 r값이 비교되어야 한다. 만약 r값이 텍셀의 값보다 크면 그 위치와 광원 사이에 다른 물체가 있다는 뜻이므로 그림자가 지게 되는 것이다. 이에 대한 대부분의 과정은 OpenGL 1.1에서 대부분 구현할 수 있지만, 텍셀 값과 r값을 비교하는 기능은 몇몇 SGI 장비를 제외하고는 아직 일반적으로 지원되지 않고 있다. 따라서, 이 논문에서는 그림자가 지게 될 물체에만 텍스춰를 붙이게 함으로써 그림자를 지게 하였는데, 이 방법으로는 물체 자체에는 그림자가 지게 하지 못하는 단점이 있다.

2.3 영상 기반 렌더링을 이용한 그림자 지도 기법

텍스춰 매핑을 이용한 그림자의 생성 기법은 그림자 지도만 미리 만들어 놓으면 매우 빠른 속도로 그림자를 생성해 줄 수 있다. 하지만 광원의 위치가 바뀌거나 물체들이 움직이는 경우에는 그림자 지도를 새로 만들어야 하기 때문에, 렌더링 속도가 급격히 느려지는 단점이 있다. 하지만 이러한 단점은 물체가 움직일 수 있는 위치에 대해 미리 그림자 지도를 생성해 놓고, 실제 렌더링을 할 때에는 물체가 움직인 위치에 해당되는 그림자 지도를 가져와서 사용하기만 하는 방법으로 해결할 수 있다[4].

2.3.1 영상 기반 렌더링의 응용

영상 기반 렌더링을 이용하면 물체나 광원이 움직이는 경우에 속도가 현저히 느려지는 기존의 방법의 문제점을 해결할 수 있다[4]. 이는 물체가 움직일 수 있는 모든 위치에 대하여 그림자 지도를 미리 생성해 놓고, 실제 렌더링에서는 물체의 위치에 따라 해당되는 그림자 지도만을 가져다 그리는 방법이라 할 수 있다. 이러한 방법을 이용하면 영상 기반 렌더링의 장점과 같이 물체나 전경의 복잡도와 상관없이 빠른 시간 안에 그림자를 생성해 낼 수 있다. 그림 4는 물체의 각 위치에 따라 생성된 그림자 지도들이다.

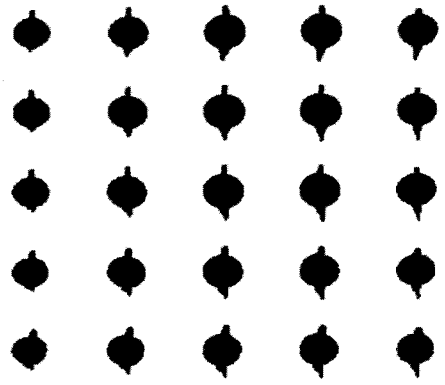


그림 4 물체의 위치에 따라 만들어진 그림자 지도

이렇게 만들어진 그림자 지도들에서 물체의 위치에 따라 해당하는 그림자 지도만 읽어와서 투영 텍스춰 기법을 이용해 그림자를 생성하면 된다. 즉, 물체의 위치가 (x, y)에 있다면 그림자 지도도 물체가 (x, y)일 때 생성된 것을 가져와서 사용하면 된다는 것이다. 그리

고, 광원이 움직이는 경우에도 빠르게 그림자를 생성할 수 있다. 광원이 움직이면 앞의 그림자 지도도 새로 만들어야 할 것 같지만, 실제로는 광원이 움직인다면, 그림자의 모양은 광원이 고정되어 있고, 물체가 반대 방향으로 움직인 경우와 그 모양이 같다. 그렇기 때문에 기존의 그림자 지도를 새로 만들지 않고, 물체가 움직인 위치만 반대 방향으로 계산해서 사용하면 되는 것이다.

2.3.2 영상 기반 렌더링을 이용한 방법의 문제점

이 방법으로는 물체나 광원이 고정되거나 움직임에 상관없이 텍스춰 매핑 하드웨어를 이용하여 빠른 속도로 그림자를 생성해 낼 수 있다. 그러나 이 방법에서는 물체가 움직일 수 있는 모든 위치에 대한 그림자 지도를 만드는 것이 아니라, 대략적으로 어느 정도의 위치에 대해서만 그림자 지도를 만들게 된다. 그렇기 때문에 물체가 그림자 지도를 만들지 않은 위치로 움직이는 경우에는 문제가 생기게 된다. 따라서 [4]에서는 이 문제점을 물체와 가장 가까운 곳의 그림자 지도를 사용하거나, 인접한 네 개의 그림자 지도를 보간하는 방법을 사용하였다. 그러나 가장 가까운 곳의 그림자 지도를 사용할 때에는 그림자가 변하는 과정이 부드럽지 못하고 끊어지는 단점이 있고, 보간을 하는 경우에는 그림자의 모양이 흐리게 되는 문제점이 있다. 또한, 물체가 위치할 수 있는 위치의 개수, 즉 그림자 지도를 만들어 낼 위치와 그림자 지도의 크기에 따라 만들어지는 그림자 텍스춰의 크기가 달라지는데, 그 크기가 매우 크다는 문제점이 있다. 만약 물체의 위치를 32×32 에 대해서 샘플링 하고, 그림자 지도의 크기를 128×128이라고 한다면, 그림자 지도를 모두 만들었을 때의 텍스춰 크기는 총 16MBytes정도가 된다. 그림자 지도의 크기가 256×256일 때에는 64MBytes, 512×512일 때에는 256MBytes라는 많은 용량을 차지하게 된다. 최근의 3차원 그래픽스 가속기들이 예전보다 많은 양의 메모리를 제공하기는 하지만, 일반적인 범용 컴퓨터의 경우 이러한 가속기들이 지원하는 메모리의 양은 많아야 32MBytes에서 64MBytes에 불과하다. 또한 이 메모리의 양도 프레임 메모리와 공유하는 것이기 때문에 실제 텍스춰 메모리로 사용할 수 있는 양은 매우 적은 실정이다. 실제로 이 방법에서는 텍스춰 메모리의 양보다 적은 크기의 그림자 지도를 사용할 경우 매우 빠른 속도로 그림자를 만들어 낼 수 있지만, 그림자 지도의 크기가 텍스춰 메모리의 양보다 커지는 경우 그 속도가 현저히 느려지는 단점이 있었다. 이러한 단점을 해결하기 위해서 빠른 디코딩 속도와 랜덤 액세스를 제공하는 Haar기저의 웨이블릿 압축 방법을 사용하였는데, 이 방법으로 지나치게 커진 그림자 지도의 크기를 매우 작게 압축할 수 있으므로 위의 문제를 해결할 수 있지만, 압축을 풀기 위해 어느 정도의 시간이 필요하기 때문에 속도의 저하는 피할 수 없게 된다.

3. 이미지 와핑을 이용한 그림자 생성

이 장에서는 기존의 영상 기반 렌더링을 이용한 방법이 지나치게 큰 그림자 지도를 필요로 한다는 단점을 해결하기 위하여 이미지 와핑 기법을 이용한 방법을 제안하고자 한다. 이미지 와핑은 주어진 이미지를 사용자가 원하는 모양으로 변형시키는 기법으로써, 이미 수많은 영화들에 특수 효과로 사용된 바 있는 방법이다. 이 방법으로 기존의 영상 기반 렌더링을 이용한 그림자 지도 기법에서 물체의 위치에 따라 많은 수의 그림자 지도를 만들어야 하는 문제점을 해결할 수 있다. 이 방법을 그림자 생성에 이용할 수 있는 이유는, 그림 5에서 보는 바와 같이 일직선으로 움직이는 물체의 그림자는 점진적으로 변한다는 사실 때문이다. 따라서 물체의 직선 움직임에 대하여 시작하는 위치의 모양과 끝나는 위치의 모양을 안다면 중간의 변화를 선형적으로 계산하여 실제의 그림자와 유사한 모양을 만들어낼 수 있다. 그러한 점진적인 변형을 이미지 와핑을 이용하여 얻어낼 수 있다는 점을 이용하여 중간 단계의 그림자를 얻어낼 수 있다는 것이다.

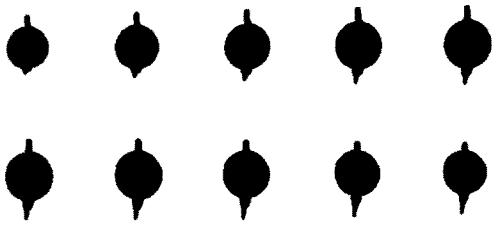


그림 5 물체의 직선적인 움직임에 대한 그림자의 변화

3.1 이미지 와핑의 소개

Wolberg는 [9]에서 이미지 와핑(image warping)은 주어진 이미지의 기하학적 변형을 다루는 분야라고 정의하였다. 한 이미지의 기하학적인 변형은 그 이미지 안에서 각 픽셀들 사이의 공간적인 관계를 재구성하는 작업이며, 두 이미지 사이에서 순차적인 변형 과정을 만들어 내는 작업을 이미지 모핑(image morphing)이라고 한다. 와핑이 이미지가 변화하는 중간 과정이 없다는 점을 제외하면 와핑(warping)과 모핑(morphing)은 거의 같은 작업이라 할 수 있다. Beier는 원래 이미지와 변형된 후의 이미지에서 그 특징을 잘 나타내주는 여러 개의 벡터들을 정의하고, 각각의 벡터들의 쌍이 어떻게 변화되어 매핑 되는가를 수학적으로 계산하여 주변의 픽셀들을 이동시키고 보간함으로써 이미지 모핑을 구현하였다[10].

3.2 이미지 와핑의 그림자 지도 기법에의 응용

앞에서도 언급했듯이, 물체가 한 지점에서 다른 지점으로 직선적으로 움직일 때, 그 그림자의 모양은 점진적으로 변화함을 알 수 있다. 물체가 움직인 첫 시작 지점과 끝난 지점의 위치를 알고, 그림자를 그리고자 하는 위치가 그 사이에서 어느 정도의 위치에 있는지 알 수 있다면, 그 비율만큼 이미지를 와핑해서 중간 단계의 그림자 모양을 얻어낼 수 있다.

Chen 등은 [18]에서 뷰 보간(view interpolation) 기법을 이용하여 면적을 가지는 광원에 의해 생기는 부드러운 그림자를 이미지 모핑을 이용해 구현하였다. 이 논문에서는 면적의 양 끝점에서 생성된 그림자 지도에서 이미지 모핑을 이용하여 그 중간 단계에 해당되는 그림자 지도들을 생성하고 누적하여 부드러운 그림자를 만들 수 있다고 하였다. 그러나 물체가 움직이는 경우에 대하여 매번 그림자 지도를 새로 생성해야 하고, 다시 모핑을 해야하기 때문에 3차원 게임과 같은 실시간 응용에 적용하기가 적합하지 않다.

이 논문에서 제안하는 방법의 궁극적인 목표는 그림자 지도를 사용하는 기존의 영상 기반 렌더링 기법보다 적은 양의 메모리를 사용하면서 대등한 속도를 얻을 수 있는 방법을 개발하는 것이다. 이를 위해서 [9]에서 설명한 메쉬 와핑 알고리즘을 이용하면 지나치게 많은 계산량을 필요로 하고 따라서 그 속도가 느려지기 때문에 적합한 방법이라 할 수 없다. 따라서 매우 빠른 속도로 이미지 와핑을 할 수 있는 방법을 필요로 하게 되었는데, 텍스춰 매핑 하드웨어를 이용한 이미지 와핑 방법이 가장 빠른 속도를 낼 수 있었다.

3.2.1 텍스춰 매핑을 이용한 이미지 와핑

기존의 메쉬 와핑 알고리즘이나 MFFD(Multilevel Free-Form Deformation)방법에 기반한 모핑으로는 한 장의 그림자를 만들기 위해서 세 번 와핑을 해야하는 본 논문의 방법에 만족할만한 속도를 내지 못한다. 그러나 텍스춰 매핑을 이용하여 이미지 와핑을 하면 매우 빠른 속도로 결과 이미지를 얻을 수 있다[11]. 텍스춰 매핑을 이용한 이미지 와핑은 다음과 같은 방법으로 구현할 수 있다. 우선, 원래 이미지의 특징을 가장 잘 나타낼 수 있는 지점들을 선택해서 정방형의 배열로 저장한다. 이 점들의 배열의 크기는 원래 이미지의 것과 결과 이미지의 것이 일치해야 하며, 각각의 점들은 일대일로 대응된다. 이렇게 원래 이미지와 결과 이미지, 그리고 두 이미지의 조절점들의 배열이 주어지면, 원래 이미지를 텍스춰 메모리로 로드한다. 그리고 물체의 위치가 두 샘플링 지점 사이에 어느 정도의 비율로 위치하는지를 계산하여, 두 조절점들의 배열을 각각 선형 보간한다. 그리고 프레임 버퍼에 이렇게 보간된 조절점들로 이루어진 삼각형들을 그리는데, 각 점들에 대하여 매핑되는 텍스춰 좌표는 원래 이미지의 조절점의 좌표로 주고 꼭지점들의 좌표는 보간된 좌표로 한다. 이렇게 다각형 메쉬를 그리게 되면 원래 이미지의 조절점들로 이루어진 삼각형들과 그 부분에 해당되는 이미지의 부분이 보간되어 이동된 점들로 이동되고 변형되게 된다. 이런 방식으로 전체 조절점들의 다각형 메쉬를 그리게 되면 와핑된 이미지를 얻게 된다. 그림 6은 이러한 와핑 과정을 보여준다.

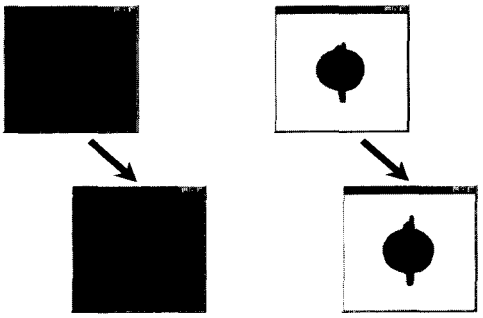


그림 6 텍스춰 매핑을 이용한 이미지 와핑 기법

텍스춰 매핑 방식을 이용한 이미지 와핑 기법이 매우 빠른 속도를 제공하기는 하지만, 기본적으로 와핑된 그림자의 모양이 실제 렌더링한 그림자의 모양과 어느 정도까지 비슷한 모양을 얻을 수 있는지도 중요한 문제이다. 아래 그림 7은 텍스춰 매핑을 이용하여 얻은 그림자의 모양과, 실제 렌더링을 통해 얻은 그림자의 모양을 비교한 것을 보여준다. 이 그림에서 볼 수 있듯이 이미지 와핑을 이용하여 만든 그림자의 모양이 실제 렌더링한 그림자의 모양과 매우 흡사함을 알 수 있다.

그림 7 텍스춰 매핑을 이용해 와핑한 그림자와 실제 렌더링한 그림자의 비교

3.2.2 그림자 지도 기법에의 적용

이미지 와핑을 이용해 중간 단계의 그림자 모양을 얻을 수 있고, 그 모양이 실제 전경에 적용되더라도 괜찮을 만큼 유사한 모양을 유지할 수 있다면, 중간 위치의 그림자 지도들을 모두 만들 필요 없이 모든 위치의 그

림자를 그릴 수 있다. 텍스춰 매핑을 이용한 와핑 방법은 그 수행 속도가 매우 빠르고 조절점들을 잘 설정한다면 상당히 괜찮은 수준의 결과를 얻을 수 있기 때문에, 본 논문에서 제안하는 방법에 적합하다 할 수 있다. 이미지 와핑을 이용해서 그림자를 생성하려면 다음과 같은 과정을 수행해야 한다.

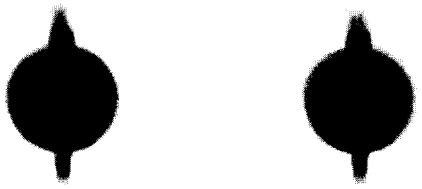
우선 설명의 편의상 물체가 평면상에서 이동한다고 가정하자. 이러한 경우 움직이는 영역에 대하여 각 모서리와 중간 지점에 대하여 3×3의 위치에 대해 미리 그림자 지도를 생성한다. 여기에서의 중간 지점은 가로 방향은 광원의 가로 방향 좌표, 세로 방향은 광원의 세로 방향 좌표에 해당되는 위치를 의미한다. 그리고 각 그림자 지도에 대한 조절점들의 배열을 저장한다. 이 두 가지 정보를 전처리 과정에서 구해 놓고 렌더링을 한다. 실제로 렌더링을 할 때 그림자를 구하는 방법은 우선, 물체의 위치에 따라 주변 4개의 그림자 지도를 선택한다. 예를 들어, 물체가 전체 범위에 대해서 중간보다 위쪽에 있고, 왼쪽에 있다면, 그림자 지도는 왼쪽 위, 왼쪽 중간, 위쪽 중간, 그리고 가운데의 4개의 그림자 지도를 선택하면 되는 것이다. 이 네 개의 그림자 지도들을 이용해서 물체가 위치한 곳의 세로 방향의 비율과 가로 방향의 비율에 대해 2차원 보간과 같은 와핑을 하면 그 지점에 대한 와핑된 그림자 지도를 얻을 수 있다.

이렇게 얻어진 그림자 지도를 기존의 방법과 동일하게 투영 텍스춰 기법을 이용해서 전경에 투영시키면 그 물체에 대한 그림자가 진 전경을 얻을 수 있다. 이런 과정으로 점 광원에 대한 경계가 뚜렷한 그림자를 얻기 위해서는 한번에 총 3차례의 와핑이 필요하게 된다.

3.2.3 부드러운 그림자로의 확장

이 논문의 앞부분에서도 언급했었지만, 실제 세계에서 점광원에 의한 경계가 뚜렷한 그림자가 생기는 경우는 흔치 않다. 실제로는 대부분의 상황에서 그림자는 면적을 갖는 광원에 의해 생기는 부드러운 그림자가 대부분이다. 그렇기 때문에 좀 더 사실적인 장면을 표현하기 위해서는 이러한 부드러운 그림자의 생성이 필수적이다. 그러나, 현존하는 그래픽스 가속기나 여러 가지 그림자 생성 알고리즘들 중에서는 이러한 부드러운 그림자를 실시간으로 생성해 줄 수 있는 방법이 없다. Heckbert 등은 면적을 가지는 광원을 여러 개의 점광원으로 간략화하고 샘플링하여 각 점광원에 대한 그림자들을 만든 다음 누적시키는 방법으로 부드러운 그림자를 생성하였다[6]. 그러나 이 방법은 광원을 많은 수의 점광원으로 나누면 그림자가 더 부드러워 지지만 그만큼 많은 그림자를 생성해야 하기 때문에 속도가 느려지고, 점광원의 개수를 줄이면 그림자가 알리아싱과 같은 계단현상이 생기는 문제점이 있다. Soler 등은 이러한 문제점을 해결하기 위해 여러 번의 샘플링 대신에 콘볼루션을 이용해 그림자를 부드럽게 만들었다[7]. Wolfgang 등은 그림자의 부드러운 부분의 광도의 변화량을 가시성 지도로 만들고, 이러한 가시성 지도를 이용하여 적은 횟수의 샘플링 만으로 부드럽게 변화하는 그림자를 표현하였다[8].

Heckbert 등이 제안한 부드러운 그림자 생성 기법은 각 샘플링 위치별로 그림자 지도만 만들어지면 누적 버퍼를 이용하여 빠르게 누적시켜 부드러운 이미지를 만들어 낼 수 있다. 그러나 대부분의 중, 저가 그래픽스 하드웨어에서는 이러한 누적 버퍼를 제공하지 않기 때문에 이 방법을 적용하기 어렵다. 그러나 대부분의 하드웨어에서 일반적으로 제공하는 알파 버퍼(alpha buffer)를 이용하여 이와 같은 작업을 수행할 수 있다. 즉, 샘플링된 모든 그림자 지도의 이미지에 알파 값을 $\frac{1}{(\text{총 광원 샘플링 횟수})}$로 넣어주고, OpenGL 블렌딩 함수에 대하여 source에 GL_SRC_ALPHA, 그리고 destination에 GL_ONE으로 준 다음 합성하면 누적 버퍼를 이용하여 누적시킨 것과 같은 결과를 얻을 수 있다. 그림 8은 이런 방법으로 여러 개의 그림자를 누적시킨 후 얻은 부드러운 그림자의 모양이다.



5개의 Sampling 에 의한 Softshadow　　9개의 Sampling 에 의한 Softshadow

그림 8 알파 버퍼를 이용하여 누적시킨 부드러운 그림자

이러한 방법에 사용될 샘플링된 그림자들은 그때마다 만들 필요가 없고, 광원의 중심 위치에서 그 샘플링 위치만큼 반대 방향으로 물체가 움직인 경우로 생각하여, 미리 만들어 놓은 그림자 지도를 사용할 수 있다. 이 방법에서는 샘플링 횟수만큼의 그림자들을 만들어 내야 하는 추가적인 와핑 과정이 필요하고, 알파 블렌딩을 사용하는 경우 알파 채널에 대한 부가적인 텍스춰 용량이 필요하게 된다. 그러나 누적 버퍼를 지원하는 하드웨어의 경우에는 이러한 알파 채널이 불필요하게 되므로, 텍스춰 메모리에 대한 추가적인 부담은 없다.

4. 구현 및 결과

4.1 구현 환경

이 논문에서 제안한 이미지 와핑을 이용한 그림자 생성 기법은 Silicon Graphics의 Visual Workstation 320

에서 구현되었다. 이 시스템은 두 개의 Intel Penti um II 450MHz의 CPU와 164MBytes의 메인 메모리, 92MBytes의 그래픽 메모리를 가지고 있으며, Silicon Graphics의 Cobalt 칩셋을 사용한 그래픽스 가속기를 사용한다. 프로그램은 Visual C++ 6.0과 OpenGL 1.1의 환경에서 구현되었다.

본 실험에서는 그림자를 생성하는 시간과 물체의 복잡도 사이의 연관성을 알아보기 위해 세 가지의 다면체 모델(주전자 : 다각형 1152개, 의자 : 다각형 1464개, 소 : 다각형 5804)로 실험하고 그 결과를 비교하였다. 또한 그림자를 생성하는데 영향을 미치는 여러 가지 요소들에 대하여 실험하였다. 실제로 그림자를 생성해내는데 영향을 미치는 요소는 그림자 지도의 크기, 와핑을 위한 조절점의 수, 부드러운 그림자를 만들기 위한 광원의 샘플링 횟수 등이 있다. 그리고, 이 논문에서 제안한 방법이 기반을 둔 영상 기반 렌더링을 이용한 그림자 생성 방법[4]의 성능과 비교하기 위하여 같은 조건하에서의 그림자 생성 시간을 비교하였다.

4.2 실험 결과

이 논문에서 제안한 그림자 생성 방법에서 한 개의 그림자의 생성에 시간적으로 가장 큰 영향을 미치는 요소로 그림자 텍스춰의 크기와 이미지 와핑에 걸리는 시간을 들 수가 있다. 반면, 본 방법에서는 물체의 그림자를 미리 만들어 놓기 때문에, 실제 렌더링 시에 그림자를 생성하는 시간은 물체의 복잡도와 관계없이 일정한 속도로 그림자를 만들어 낼 수 있다. 표 1은 서로 다른 복잡도를 가지는 물체들에 대하여 서로 다른 크기, 즉 64 × 64에서 512 × 512까지의 크기를 가지는 그림자 텍스춰의 와핑을 통하여 실제로 투영을 할 그림자를 생성하는데 걸리는 시간을 비교하고 있다.

이 표에서 '그림자'는 전체 렌더링 시간('전체') 중 와핑을 통한 그림자 생성에 걸리는 시간으로서 본 논문에서 제안하는 방법에서는 그림자 생성 속도는 물체의 복잡도보다는 그림자 지도의 크기에 더 크게 영향을 받음을 알 수가 있다. 따라서 이미지 와핑을 이용한 그림자 지도 기법에서는 전경의 복잡도와 상관없이 거의 균일한 속도로 그림자를 생성해 낼 수 있다.

표 1 물체의 복잡도에 따른 그림자 생성 속도
(단위: ms, 조절점 개수: 5×15)

물체	그림자 지도의 크기							
	64 × 64		128 × 128		256 × 256		512 × 512	
	그림자	전체	그림자	전체	그림자	전체	그림자	전체
주전자	4	12	5	14	8	19	13	55
의자	5	15	6	18	8	25	13	59
소	5	23	6	29	9	31	14	68

표 2 그림자 텍스춰의 크기에 따른 렌더링 시간
(단위: ms, 조절점 개수: 5×15)

방 법	그림자 지도의 크기			
	64 × 64	128 × 128	256 × 256	512 × 512
직접 렌더링	22	48	87	204
와핑 방법	4	5	8	13

다음의 표 2는 주전자 물체에 대하여 그림자 텍스춰의 크기에 따른 그림자 생성 속도를 비교한 결과이다. 여기서 '와핑 방법'은 본 논문에서 제안하는 방법을 사용하는 것이고, '직접 렌더링' 방법은 그림자 생성을 위하여 시점을 광원에 놓고 물체를 렌더링한 후 깊이 버퍼의 내용을 읽어 들여 텍스춰 형태의 그림자로 생성을 하는데 걸리는 시간이다. 당연한 결과이겠지만 와핑 방법이 훨씬 빠르게 그림자를 생성함을 알 수가 있다.

그림자 생성, 특히 이미지 와핑에 걸리는 시간은 그림

표 3 조절점의 개수에 따른 렌더링 시간의 비교 (단위 : ms)

조절점 갯수	그림자 종류	그림자 지도의 크기			
		64 × 64	128 × 128	256 × 256	512 × 512
7 × 15	Hard Shadow	4	5	8	15
5 × 15	Hard Shadow	4	5	8	13
7 × 15	Soft Shadow (Sampling = 5)	20	21	40	80
5 × 15	Soft Shadow (Sampling = 5)	19	21	38	79
7 × 15	Soft Shadow (Sampling = 9)	33	41	60	140
5 × 15	Soft Shadow (Sampling = 9)	33	40	58	142

자 텍스춰의 크기 외에도 조절점의 개수에도 영향을 받는다. 조절점의 개수에 따라 CPU에 의존하는 보간 계산의 양이 달라지기 때문에 조절점의 개수가 많아질수록 와핑에 걸리는 시간이 느려진다고 할 수 있다. 표 3은 조절점의 개수에 따른 그림자의 생성 속도를 비교한 것인데, 여기서 'Hard Shadow'는 주전자 물체에 대하여 이 논문에서 제안한 이미지 와핑을 이용한 방법으로 뚜렷한 그림자를 만드는 시간이고, 'Soft Shadow'는 각각 다섯 번과 아홉 번의 샘플링을 통하여 부드러운 그림자를 만드는 데 걸리는 시간을 의미한다. 여기서 나타난 결과에서 보듯이 그림자 텍스춰의 해상도가 커짐에 따라 그림자 생성 시간이 길어짐을 알 수 있는데, 이는 텍스춰 매핑을 이용하여 와핑한 그림자를 프레임 버퍼에서 읽어들여 텍스춰 메모리로 읽어 들이는 데 걸리는 시간이 해상도에 따라 달라지기 때문이다. 실제로 이 와핑 방법에서 그림자를 생성해 내는데 CPU가 부담하는 계산은 각 조절점들을 선형 보간하는 연산 정도이고, 나머지 부분은 텍스춰 매핑 하드웨어를 이용하여 수행하기 때문에, 그림자를 생성하는데 걸리는 많은 부분의 시간은 프레임 버퍼와 텍스춰 버퍼 사이에 읽고 쓰는 시간이라 할 수 있다. 또한 조절점의 크기에 따라 CPU가 부담하게 되는 계산량이 달라지는데, 그 계산량이 매우 적기 때문에 실제로 그림자 생성 시간에 미치는 영향은 그리 크지 않다. 이러한 사실은 표 3에서 보듯이 조절점의 개수가 변해도 그림자 생성 시간은 크게 변하지 않음에서 알 수 있다.

표 4는 이 논문에서 제안한 방법과 기존의 영상 기반 렌더링 기법을 이용한 그림자 생성 방법[4]의 성능을 비교한 것이다. 각 실험은 다양한 그림자 지도의 크기에 대하여 경계가 뚜렷한 그림자를 생성하는 속도를 측정한 것이다. 기존의 방법은 그림자 텍스춰들을 압축하지 않고 사용하는 경우('기존의 방법-비압축')와 [23]에서 제시한 웨이블릿 기법으로 압축하여 사용하는 경우('기존의 방법-압축')에 대하여 실험하였다. 압축을 하는 경우에 실제로 텍스춰를 디코딩해야 하는 시간이 추가되므로 압축을 하지 않을 경우보다 느리게 된다. 참고로 본 실험에서 512×512 해상도의 그림자 텍스춰를 사용할 경우 압축을 하지 않을 때 전체 그림자 데이타의 크기가 하드웨어에서 제공하는 텍스춰 메모리보다 많아져 메인 메모리와의 스워핑이 자주 일어나 느린 결과를 얻었다. 전체적으로 보면 [4]의 방법에서 압축을 할 경우 본 논문에서 제안한 방법과 비슷한 그림자 생성 속도를 보여 주고 있다. 일반적으로 압축을 할 때 장면이 복잡해질 경우 필요로 하는 그림자 텍스춰의 크기가 와핑 방법에 기반한 방법과 비교할 경우보다 훨씬 커지게 된다. [4]의 방법에서는 네 개의 주변 텍스춰에 대한 보간을 통하여 중간 위치에 대한 그림자를 생성하는데, 이러한 경우 보통 그림자가 부예지며 그림자의 형태가 일그러지는 반면, 본 논문에서 제안하는 방법에서는 와핑을 통하여 뚜렷한 형태를 가지는 그림자를 생성할 수가 있다. 또한 [4]의 방법과 이 방법과의 특징을 비교하면 전자의 경우 그림자 생성에 필요한 계산의 대부분이 CPU에서 수행이 되는 반면, 후자의 경우 대부분의 계산이 그래픽스 가속기에 의해 수행이 된다는 점이다.

마지막으로 표 5는 부드러운 그림자에 대한 새로운 방법의 그림자 생성 속도를 측정한 결과이다. 모든 그림자를 직접 렌더링하여 매번 새롭게 만드는 기존의 방법과 이미지 와핑을 이용한 새로운 방법의 그림자 생성 속도를 비교하였다. 이 결과에서 나타난 부드러운 그림자를 생성하는 속도는 충분히 실시간 렌더링을 가능하게 하는 성능이며, 이미지 압축에 기반하는 기존의 방법보다 매우 적은 양의 메모리를 사용하기 때문에 더욱 복잡한 전경에 대한 적용이 가능하다고 할 수 있다. 또한 표 5에서 보듯이 새로운 방법은 부드러운 그림자를

표 4 [4]에서 사용되었던 방법과 새로운 방법의 성능 비교 (단위 : ms, 조절점 개수: 5×15)

그림자 지도의 크기	기존의 압축을 통한 방법[4]		이미지 와핑을 이용한 방법
	비압축	압축	
64 × 64	4	5	4
128 × 128	6	8	5
256 × 256	6	11	8
512 × 512	30	25	13

표 5 부드러운 그림자에 대한 새로운 방법의 성능 분석 (단위 : ms, 조절점 개수: 5×15)

그림자 지도의 크기	직접 렌더링		새로운 방법	
	(5번 샘플링)	(9번 샘플링)	(5번 샘플링)	(9번 샘플링)
64 × 64	161	198	19	33
128 × 128	204	483	21	40
256 × 256	472	774	38	58
512 × 512	1075	1936	79	142

그림 9 경계가 뚜렷한 그림자 결과

그림 10 부드러운 그림자 결과

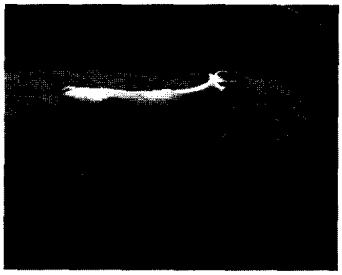


그림 12 소에 대한 부드러운 그림자

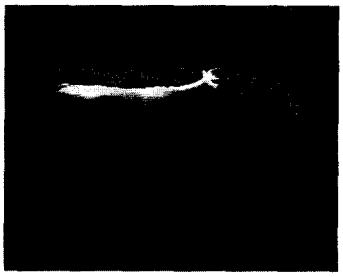


그림 11 소에 대한 경계가 뚜렷한 그림자

만드는 데 실시간 렌더링이 전혀 불가능할 정도로 느린 기존의 방법과 비교하여 최고 15배 가량 빠른 속도를 보여주었다.

그림 9-12는 기존의 경계가 뚜렷한 그림자를 사용하여 렌더링 한 결과와 부드러운 그림자를 통하여 생성한 렌더링 결과를 보여주고 있는데, 후자가 훨씬 자연스러운 모습을 표현하고 있음을 알 수가 있다.

5. 결 론

최근의 그래픽스 하드웨어의 급격한 발달로 인해, 3차원 컴퓨터 그래픽스 분야에서 사실적인 영상을 실시간으로 생성해 내는 것이 매우 중요한 요소로 대두되고 있다. 여기에는 사실적인 그림자의 생성 방법도 포함된다. 현재 수많은 알고리즘들과 그래픽스 하드웨어들이 나타났지만, 아직도 사실적인 그림자를 실시간으로 생성해내는 것이 용이하지 않다. 그 중 영상 기반 렌더링 기법을 이용한 그림자 생성 방법은 실시간으로 그림자를 생성해 내기에 충분한 성능을 제공하지만, 알고리즘의 특성상 매우 많은 양의 텍스춰 메모리를 사용해야 하는 것이 단점이라 할 수 있다.

본 논문에서는 이러한 단점을 해결하고자 기존의 영상 기반 렌더링을 이용한 그림자 생성 기법에 이미지 와핑 기법을 도입하여 그림자 지도의 크기를 대폭 줄일 수 있는 방법을 제안하였다. 이 방법에서는 텍스춰 매핑 기법을 이용하여 이미지 와핑을 매우 빠른 속도로 수행할 수 있었으며, 그로 인해 실제 세계에서와 같은 부드러운 그림자를 실시간에 가까운 속도로 생성해 낼 수 있었다. 또한, 이미지 와핑을 했을 때, 실제 렌더링한 그림자 모양과 유사한 결과를 얻어낼 수 있을 만큼의 최소한의 그림자 지도만을 미리 생성하여 사용하기 때문에, 기존의 영상 기반 렌더링 기법을 이용한 방법보다 훨씬 적은 크기의 그림자 지도만으로 그와 비슷한 속도로 그림자를 생성해 낼 수 있었다. 또한 텍스춰 메모리에 대한 부담이 적고, 생성 속도가 빠르기 때문에 면적을 가지는 광원에 대한 부드러운 그림자를 생성하는 경우에도 매우 빠른 속도로 그림자를 생성할 수 있었으며, 물체의 다양한 움직임에 대하여 고차원의 그림자 지도 데이타를 사용하는 기법으로 확장하기에도 용이하다.

그러나 이 논문에서 제안한 방법은 현재 각 그림자 지도에 대한 조절점을 수동으로 선택해야 한다는 단점이 있다. [18]에서는 두 참조 이미지간의 매핑 관계를 자동으로 설정하기 위하여 각 이미지에 해당하는 뷰잉 인자로부터 두 이미지간의 차이를 나타내는 4행 4열 변환 행렬을 구하고, 이를 통하여 오프셋 벡터로 표현된 '모프 맵'을 구하였다. 이 방법에서는 중간 단계의 이미지를 구하기 위하여 '블록 압축'이라는 방법으로 압축된 모프 맵을 사용하여 빠르게 보간 계산을 수행하였다. 이 방법은 비교적 우수한 결과를 산출하기는 하나, 본 연구

의 목적인 현존하는 그래픽스 가속기를 사용한 부드러운 그림자 생성에 적용하기가 용이하지가 않다. 본 논문의 방법에서 사용된 그림자 텍스춰는 그림자의 모양만을 가지는 이미지이기 때문에 와핑하는 방법이 간단하기는 하지만, 모든 작업을 자동적으로 수행하기 위해서는 조절점을 자동적으로 설정해 줄 수 있는 적당한 방법이 필요하다. 실험을 통하여 조절점 개수에 따른 성능을 비교해본 바로는, 조절점의 개수가 전체적인 성능에 미치는 영향은 비교적 적기 때문에, 향후 연구 과제로써 적절한 개수의 조절점을 자동으로 지정해 줄 수 있는 방법을 개발할 수 있다면 그림자 생성의 전 과정을 자동적으로 수행하여 더 좋은 결과를 얻을 수 있을 것이다. 이 논문에서 제안한 방법은 앞으로 더욱 복잡하고 정교해질 실시간 그래픽스 분야에서 자연스러운 그림자를 생성해 내는 데에 유용하게 사용될 수 있을 것이다.

참고문헌

[1] Franklin C. Crow, "Shadow Algorithms for Computer Graphics," *Computer Graphics (SIGGRAPH '77 Proceedings)*, pp.242-248, 1977.

[2] Lance Williams, "Casting Curved Shadows on Curved Surfaces," *Computer Graphics (SIGGRAPH '78 Proceedings)*, pp.270-274, 1978.

[3] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, Paul Haeberli, "Fast Shadows and Lighting Effects Using Texture Mapping," *Computer Graphics (SIGGRAPH '92 Proceedings)*, pp.249-252, 1992.

[4] 이중연, 임인성, "영상기반 렌더링 기법을 이용한 실시간 그림자 생성", 컴퓨터그래픽스학회논문지, 제7권, 제1호, pp.27-35, 2001년 3월.

[5] Hansong Zhang, "Forward Shadow Mapping," *Eurographics Workshop on Rendering '98 Proceedings*, pp.131-138, 1998.

[6] Paul S. Heckbert, Michael Herf, "Simulating Soft Shadows with Graphics Hardware," *Technical Report TR CMU-CS-97-104*, Carnegie Mellon University, 1997.

[7] Cyril Soler, Francois X. Sillion, "Fast Calculation of Soft Shadow Textures Using Convolution," *Computer Graphics (SIGGRAPH '98 Proceedings)*, pp.321-332, 1998.

[8] Wolfgang Heidrich, Stefan Brabec, Hans-Peter Seidel, "Soft Shadow Maps for Linear Lights," http://www.opengl.org/news/archives00/oct00.html, 2000.

[9] Wolberg, G., *Digital Image Warping*, IEEE Computer Society Press, 1990.

[10] Beier T, Neely S, "Feature-Based image metamorphosis," *Computer Graphics (SIGGRAPH '92 Proceedings)*, Vol 26(2), pp.35-42, 1992.

[11] Tom McReynolds, David Blythe, "Advanced Graphics Programming Techniques Using OpenGL," *SIGGRAPH '99 Course notes*, 1999.

[12] Williams T. Reeves, David H. Salesin, Robert L. Cook, "Rendering Antialiased Shadows with Depth Maps," *Computer Graphics (SIGGRAPH '87 Proceedings)*, pp.283-291, 1987.

[13] James F. Blinn, "Me and My (Fake) Shadow," *IEEE Computer Graphics & Applications*, 8(4), pp.82-86, 1988.

[14] Andrew Woo, Pierre Poulin, Alain Fournier, "A Survey of Shadow Algorithms," *IEEE Computer Graphics & Applications*, 10(6), pp.13-32, 1990.

[15] Lynne Shapiro Brotman, Norman I. Badler, "Generating Soft Shadows with a Depth Buffer Algorithm," *IEEE Computer Graphics & Applications*, 4(10), pp.5-24, 1984.

[16] Norman Chin, Steven Feiner, "Fast Object-Precision Shadow Generation for Area Light Sources Using BSP Trees," *1992 Symp. on Interactive 3D Graphics*, pp.21-30, March, 1992.

[17] George Drettakis, Eugene Fiume, "A Fast Shadow Algorithm for Area Light Sources Using Back Projection", *Computer Graphics (SIGGRAPH '94 Proceedings)*, pp.223-230, 1994.

[18] Shenchang Eric Chen, Lance Williams, "View Interpolation for Image Synthesis," *Computer Graphics (SIGGRAPH '93 Proceedings)*, pp.279 - 288, 1993

[19] Alan Watt, Mark Watt, *Advanced Animation and Rendering Techniques*, ACM Press, New York, 1992.

[20] James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes, *Computer Graphics: Principles and Practice*, 2nd Edition, Addison Wesley, Reading MA, 1996.

[21] Tom McReynolds, "Advanced Graphics Programming Techniques Using OpenGL," *SIGGRAPH '98 Course notes*, 1998.

[22] 이승원, 웨이블릿 기반 압축기법을 이용한 영상기반 렌더링의 성능 향상에 관한 연구, 석사논문, 서강대학교 컴퓨터학과, 1998년 2월.

[23] Chandrajit Bajaj, Insung Ihm, Sanghun Park, "3D RGB Image Compression for Interactive Applications," *ACM Transactions on Graphics*, Vol. 20, No. 1, pp.10-38. January 2001.

강 병 권

1999년 2월 서강대학교 컴퓨터학과 졸업(공학사). 2001년 2월 서강대학교 컴퓨터학과 졸업(공학석사). 현재 서강대학교 컴퓨터학과 박사과정 재학중. 관심분야는 볼륨 렌더링, 실시간 렌더링, Programmable Hardware 등

임 인 성

1985년 2월 서울대학교 계산통계학과 졸업(이학사). 1987년 5월 Rutgers-The State University of New Jersey 컴퓨터학과 졸업(이학석사). 1991년 7월 Purdue University 컴퓨터학과 졸업(이학박사). 1999년 7월 ~ 2000년 6월 University of Texas at Austin의 TICAM (Texas Institute of Computational and Applied Mathematics) 방문 연구원. 1993년 3월 ~ 현재 서강대학교 컴퓨터학과 교수. 관심분야는 컴퓨터 그래픽스, 과학적 가시화, 고성능 계산 등